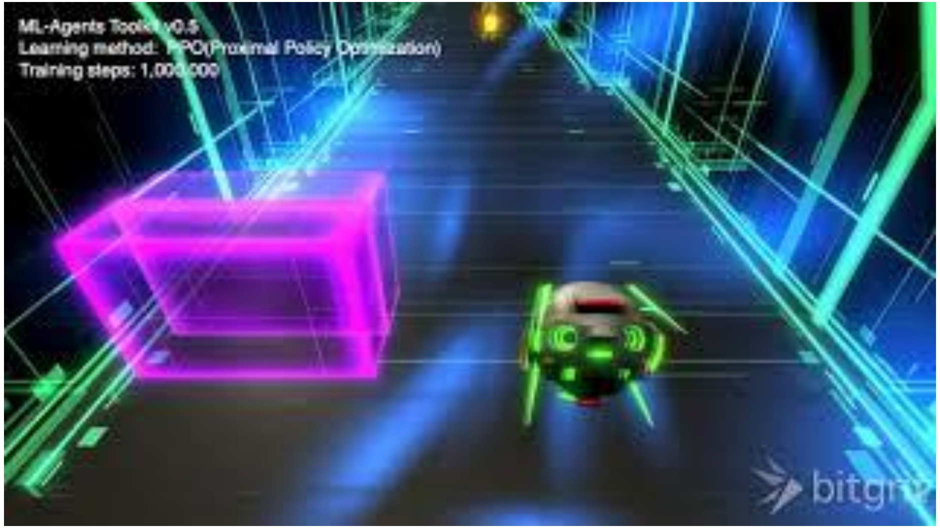
Proximal Policy Optimization

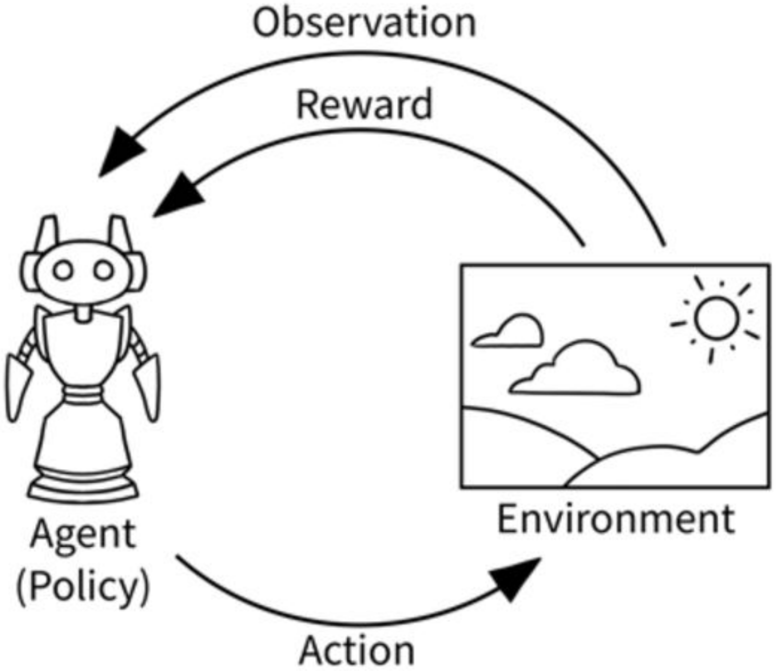
John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov

Spark interest



Agent-Environment Loop

- Agent interacts with environment through action
- Environment gives feedback (rewards and states
- Agent takes another action and the environment responds again
- This cycle repeats

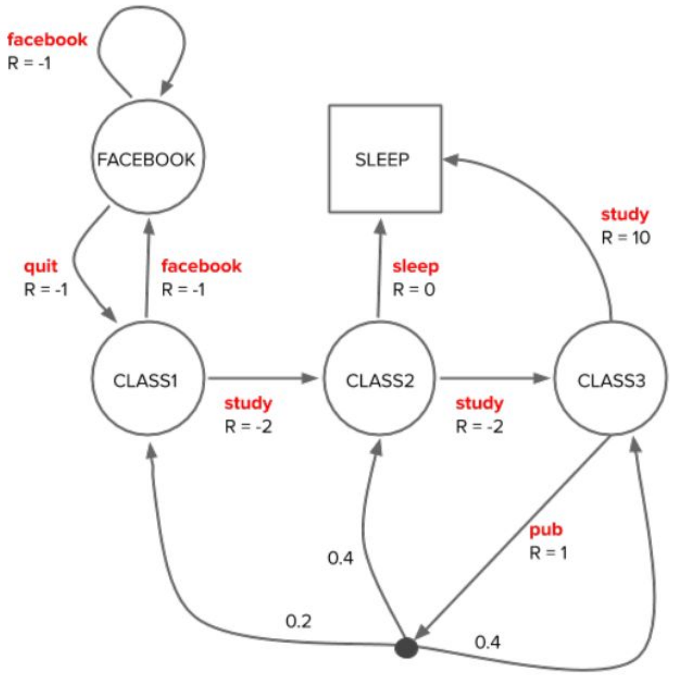


https://www.researchgate.net/profile/Adrian-Prados/publication/369550525/figure/fig1/AS:11431281130765673@1679925646365/Classical-agent-environment-loop-in-the-Reinforcement-Learning-paradigm-from-1.png

Markov Decision Process(MDP)

- Formal description of an environment
- Consists of :
  - A state space S, i.e., a set of all possible states
  - An action space A, i.e., a set of all possible actions
  - An initial state distribution $P(S\_t)$
  - A state transition distribution $P(S\_t+1|S\_t, A\_t)$
  - A reward function $R(S\_t, A\_t)$

Markov Decision Process(MDP)



Return

- Normal return function:

$$G_t = \sum_{k=0}^{\infty} R_{t+k+1}$$

- Discounted return function:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Policy

- A policy π is a distribution over actions given a state
- **Stochastic policy** : represents the probability of taking action a when in state s

$$\pi(a|s) = P(A_t = a|S_t = s)$$

- **Deterministic Policy**: function that maps each state s to a specific action a

$$\pi : S \rightarrow A \ \text{ where } a = \pi(s)$$

- Goal : maximize the expected return (cumulative reward) from any given state

Value function

- value function represents the expected return starting from state s and following policy π

:

$$V^{\pi}(s) = E^{t}[G_t | S_t = s]$$

- where G is the return (cumulative future reward) from time step t

- action-value function represents the expected return starting from state s, taking action a, and following policy π

$$Q^{t}(s,a) = E^{t}[G_t | S_t = s, A_t = a]$$

- The value function helps in evaluating the desirability of states (or state-action pairs) under a particular policy

Bellman expectation equation for value function

- Basic idea behind Bellman expectation :
  - The value of your starting point is the immediate expected reward, plus the value of wherever you land next.

$$V^{\pi}(s) = E^{\pi}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s]$$

- V^π(s) is the value of a state under policy π
- R_t+1 is the reward received after transitioning from state s to the next state S_t+1
- γ is the discount factor (0 ≤ γ <1)
- S_t+1 is the next state following policy π

$$Q^{\pi}(s, a) = E^{\pi}[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

Temporal difference learning

- combination of Monte Carlo methods and Dynamic Programming (DP).
- updates the value of a state based on the observed return and the estimated value of the next state

  $$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

- α learning rate
- Similar to Bellman equation
  - Both update the Value function
  - TDL only considers the value function
  - Bellman also considers the Reward

Policy Gradient Methods

- Define objective function that evaluates a policy

  $$J(\theta) = E_{\pi_\theta}[G_t]$$

- θ are all parameters of the neural network
- Goal: maximize the expected return
- updating the policy parameters in the direction that increases the expected return using **gradient ascent**

  $$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

- Cannot compute the gradient of an expectation

  → **Policy Gradient Theorem**

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(A_t|S_t) * G_t]$$

- log π_θ(A_t|S_t) is the logarithm of the probability of taking action A_t in state S_t according to the policy parameterized by θ
- In REINFORCE, this term scales the gradient by the return to give higher weight to actions with higher returns
- It's the 'advantage' of taking action A_t in state S_t over the average return. It indicates how much better the action was compared to the average
- By using G_t, the gradient is scaled by the advantage, guiding the optimization towards actions that lead to higher returns

Monte Carlo Estimation

- The expected value of any random variable X can be approximated by the empirical mean of independent samples x_1, … , x_n

$$\mathbb{E}[f(X)] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(A_t \mid S_t) \cdot G_t] && \text{(Policy gradient theorem)} \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(a_i \mid s_i) \cdot g_i && \text{(Monte Carlo estimation)} \end{aligned}$$

REINFORCE algorithm

REINFORCE

1. Initialize the parameters θ
2. Collect an episode $s_0, a_0, r_1, s_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$ with π_θ
3. Update the parameters for each time step t:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot g_t$$

   where $g_t = r_{t+1} + r_{t+2} + \ldots + r_T$ is the collected return

4. Go to step 2

Why computing gradients?

- The gradient indicates how to adjust policy parameters θ to improve performance.
- Using gradients, we update the policy to maximize expected returns
- Direction of Improvement
- Policy Improvement
- Policy Stability
  - By computing the gradient for each sampled trajectory and then taking the average, we obtain an estimate of the true gradient of the expected return

Trust Region Methods (TRPO)

- On-policy algorithm
- Used for environments with discrete/continuous action spaces
- Update by using largest step
- Calculate distance between policies π_old and π
- KL-Divergence

$$KL(P, Q) = \sum_{x \in X} p(x) \log\left(\frac{p(x)}{q(x)}\right)$$

- Large differences possible -> using a penalty

Advantage function

- The estimate of the relative value of the selected action
- $\hat{A}_t$ = Return R_t - Value Function (Estimate Baseline)
- Return
  - The actual rewards the agent received
- Value Function V(s)
  - s -> Current state
  - Estimate of discounted reward from this state onwards
- If positive the policy is updated, to which the behavior of the policy is reinforced
- If negative the opposite will happen

TRPO functionality

- TRPO maximizes objective function

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right].$$

$$r_t(\theta) \;=\; \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)},$$

Objective of PPO

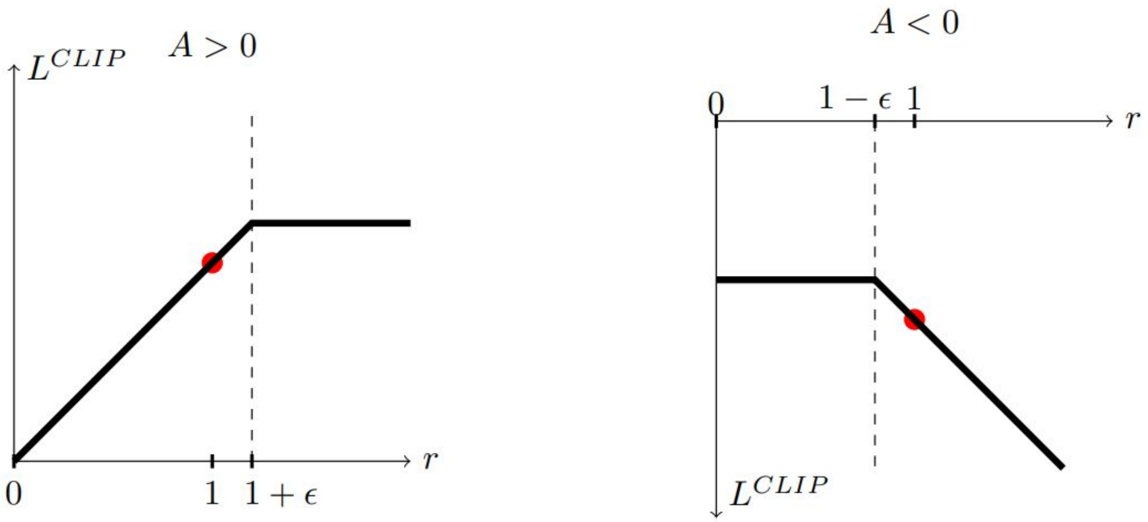
- similar to TRPO
- Penalizes changes away from 1

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- Using clipping with hyperparameter epsilon
- Min of both objectives
- Creating lower bound

Objective of PPO

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$



Schulman et al., Proximal Policy Optimization Algorithms (2017),
https://arxiv.org/pdf/1707.06347.pdf, page 3.

PPO pseudo code

Algorithm 1 PPO, Actor-Critic Style

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Schulman et al., Proximal Policy Optimization Algorithms (2017), https://arxiv.org/pdf/1707.06347.pdf, page 5.

Performance metrics

- Test using 7 robotics tasks (OpenAI Gym)
- 1 Million timesteps
- 3 Algorithms
  - No clipping
  - Clipping with parameters
  - Adaptive/Fixed KL
- Normalize results
  - 0 -> random policy , 1 -> best score

No clipping or penalty:	$L_t(\theta) = r_t(\theta)\hat{A}_t$
Clipping:	$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$
KL penalty (fixed or adaptive)	$L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

- Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$
  - If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

Performance results

- No clipping -> negative reward (half cheetah very negative)
- Clipping with hyperparameter epsilon = 0.2 best

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

Schulman et al., Proximal Policy Optimization Algorithms (2017), https://arxiv.org/pdf/1707.06347.pdf, page 6.

Results from paper

Schulman et al., Proximal Policy Optimization Algorithms (2017), https://arxiv.org/pdf/1707.06347.pdf, page 7.

How to implement PPO

- Class Network
  - Defines neural network
- Class PPO
  - Init() - Step 1
    - Step 1
  - learn() - Steps 2-7
  - rollout() - Step 3
    - Called in learn()
  - computeRewToGo() - Step 4
    - Called in rollout()
  - Along with some help functions

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: **for** $k = 0, 1, 2, \ldots$ **do**

3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

4: Compute rewards-to-go $\hat{R}_t$.

5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

8: **end for**

dortmund university

Thank you for listening